Производительность блокчейнов "по-чесноку": как измерять и не ошибаться

Прилуцкий Сергей, MixBytes.io





Кто мы?

Компания: MixBytes

Сайт: mixbytes.io

Телеграм-канал: https://t.me/mixbytes_pub

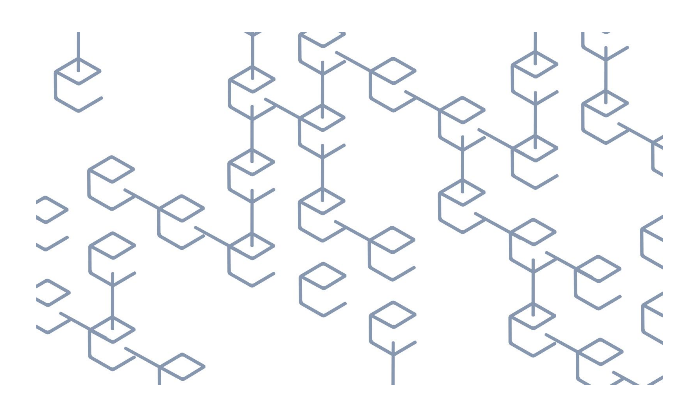
Род деятельности: разработка, аудит, исследования, консалтинг в области децентрализованных технологий

Мы — команда программистов и аудиторов кода, увлеченные децентрализованными протоколами. Мы аудируем топовые блокчейн-проекты, разрабатываем и тестируем блокчейны, консультируем компании, пишем статьи и документацию. Входим в мировой топ компаний по аудиту и разработке блокчейн-решений



Блокчейны и базы данных

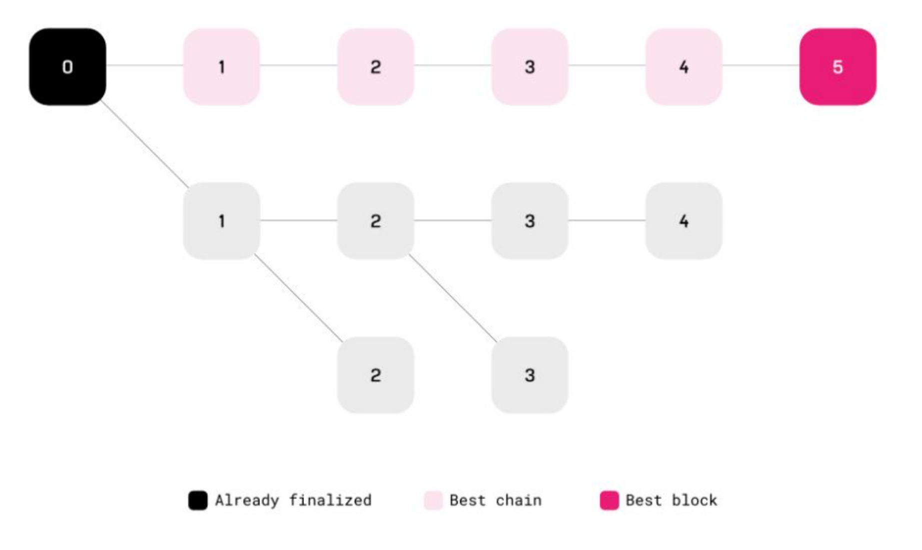
- блокчейны ~~ распределенные базы данных
- алгоритмы консенсуса ~~ репликация "master-master"
- смарт-контракты ~~ хранимые процедуры
- процессинг новых блоков ~~ работа с write-ahead log(WAL) в БД
- финализация блока ~~ commit transactions





Разрабатываем консенсус

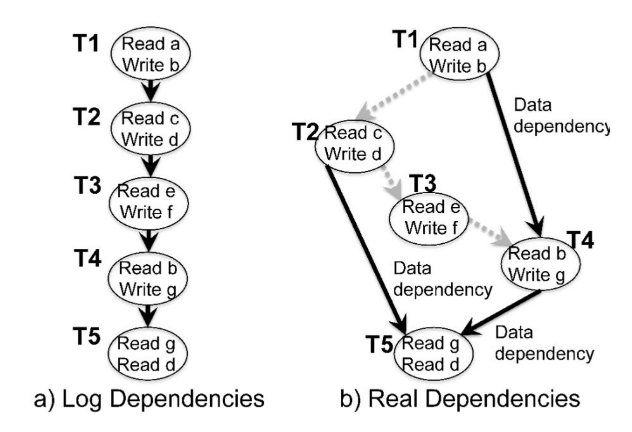
- первый слой — валидация блока, произведенного одним из валидаторов
- второй слой — финализация, асинхронный сбор подписей (тот самый BFT-консенсус)
  - + проблемы “перепроигрывания” цепочки
  - + проблемы создания snapshots
  - + проблемы отката невалидных ветвей
- итого под капотом у любого БЧ:
  - LevelDB, RocksDB + snapshots + revert logic





Тестируем логику сетевого консенсуса

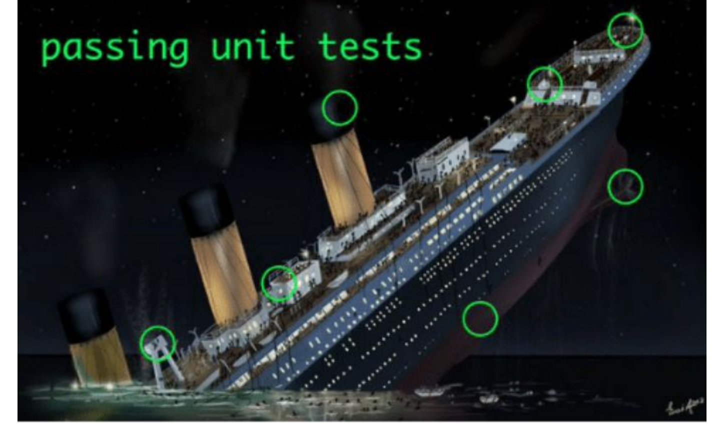
- синтетические тесты, симуляция консенсуса
- не забыть протестировать:
  - изменение списка валидаторов
  - откат невалидных ветвей
  - replay с нуля
  - восстановление из snapshot'а
- а что с эксплуатационными проблемами?





Эксплуатационные проблемы консенсуса (общие)

- традиционно для любых распределенных БД:
  - влияние сети и характера транзакций
  - нужна нагрузка и реальные блоки с транзакциями
  - разгребание pool'a транзакций
  - stateful-состояние — чем дальше, тем "тяжелее" БД





Эксплуатационные проблемы консенсуса (blockchain-specific)

- между валидаторами p2p-сеть
- доступ к валидаторам — через full-ноды
- governance может сильно изменить состав валидаторов
- из-за криптографии можно легко упереться в CPU
- реализации кода блокчейн-нод на разных языках



Что есть готового под блокчейны?

- наиболее развитые:
  - Hyperledger Caliper
    - оркестратор docker-образов
    - нет облака, эмуляции geo/packetloss, только под решения семейства Hyperledger
  - Whiteblock Genesis
    - benchmark as-a-service
    - свое облако, в котором разворачивается нужное число нод
    - заточен под Ethereum-like-сети
- есть и другие решения:
  - обычно это локальная оркестрация docker-образами



Как это было у нас

- модифицировали готовый сетевой консенсус EOS
- требования:
    - финальность за 2-5 сек
    - работоспособность сети с ~100 валидаторов, распределённых по миру
    - скачкообразное изменение числа валидаторов
- грант на тестирование блокчейн-сети Polkadot
- как протестировать, что консенсус заработает в реальной сети под нагрузкой?



Ошибки, допущенные при ручном тестировании

- тестировали на небольшом числе валидаторов
- не протестировали случай резкого увеличения числа валидаторов
- не увидели нехватку CPU на нодах
- было трудно собирать логи и coredump'ы
- не сразу сделали хороший паттерн тестирования

- нужно было решение с повторяемыми тестами, большим числом валидаторов, эмуляцией packetloss/geo и сбором логов



Какие подходы есть?

- от полностью локальных тестов к полному testnet
  - внутренние синтетические тесты
    - покрывают сложную логику
    - 100% воспроизводимы
    - но плохо описывают реальные условия
  - полный testnet с десятками нод
    - самые реальные условия
    - плохо воспроизводимы
    - дорого!
- хорошее решение находится между этими крайностями

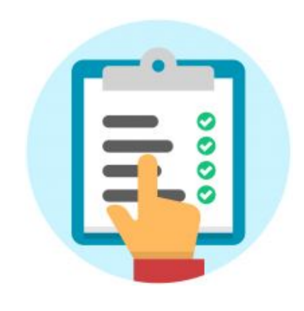


- чтобы выбрать правильно, определим требования



Требования к решению

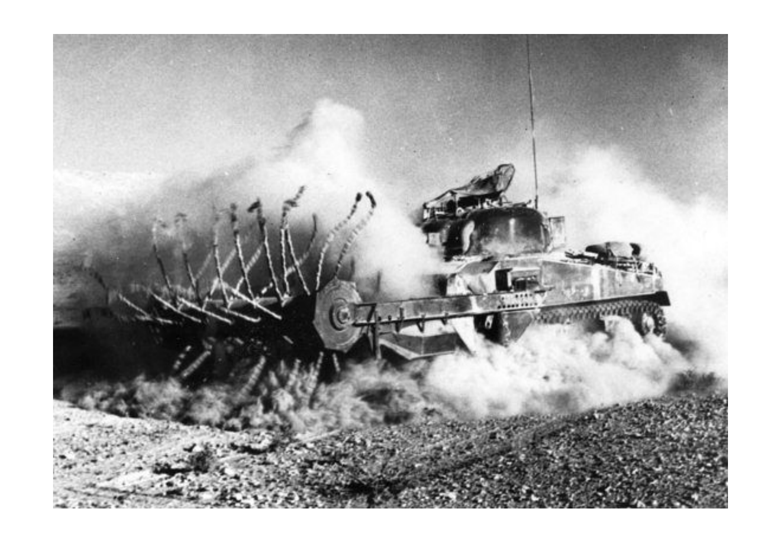
- воспроизводить конфигурацию и функционал кластера на 1,2,..., 100 машинах
- автоматически проводить setup сети (создавать много рандомных аккаунтов, выдать им крипту)
- собирать метрики с нод блокчейна и с клиентов
- запускать тесты в повторяемых условиях
- эмулировать проблемы реальных сетей (гeораспределение, packetloss)
- запускаться в разных cloud-провайдерах и делать это максимально дешево
- абстрагировать код для тестирования и бенчмаркинга от инфраструктуры запуска и сбора метрик
- иметь возможность встраиваться в процесс разработки





Так появился mixbytes [tank]

- разворачивает сети в Digital Ocean, GCE
- перед запуском теста создает с нуля нужные VPS
- после теста удаляет все использованные VPS
- эмулирует packetloss и географию
- использует многопоточную отправку транзакций с клиентов
- собирает преднастроенные метрики и позволяет добавлять свои собственные
- использует заданные разработчиком виды транзакций
- собирает логи и coredump-файлы
- позволяет добавлять новые типы блокчейнов, БД и новые облака
- на данный момент работает с Substrate/Polkadot и EOS-based-блокчейнами





Use cases

- разработчик консенсуса:
    - воспроизводимый бенчмарк сети, анализ логов в случае проблем
    - мини-кластер на своей машине
- безопасник/аудитор/аналитик
    - stress benchmark, fuzzing сети
- CI/CD:
    - автоматический бенчмарк после последнего коммита
- product owner:
    - было/стало
- и другие...





Состав решения: основные компоненты

- язык: Python 3
- Terraform: для работы с облаком
    - единый интерфейс для разных облаков (GCE, DO)
    - понимает их специфические API
    - создает новые VPS, на которых поднимаются блокчейн-ноды, бенчмарки и мониторинг
    - позволяет указать географию нод и packetloss ratio (для облаков, поддерживающих эту опцию)
    - удаляет все VPS в конце теста
- готовит inventory для ansible



Состав решения: основные компоненты

- Ansible: сценарии для запуска сети
    - использует сервера, предоставленные Terraform
    - раскладывает конфиги, генерирует ключи для каждого сервера
    - собирает со всех нод и прописывает peer'ы
    - имеет стандартные роли для prometheus + grafana
- ansible playbooks пишутся под отдельный блокчейн
    - часто уже существуют готовые
- локальное тестирование сценариев в docker'ах с помощью Molecule

```
- name: "Start {{ bc_name }}-{{ bc_component_name }} container"
  tags:
    - prod
  docker_container:
    name: "{{ bc_name }}-{{ bc_component_name }}"
    image: "{{ bc_polkadot_image }}"
    command: |
      /usr/local/bin/polkadot -d /state
      --chain=bench --validator --ws-external --rpc-external --rpc-cors=all
      --bootnodes "/ip4/{{ bc_boot_ip[0] }}/tcp/30333/p2p/QmXS53cQyDRT7RaXiKYLjfkX8xSc9pBDPohDh1F3HxzjAz"
    hostname: "{{ bc_name }}-{{ bc_component_name }}"
    network_mode: host
    volumes:
      - "{{ bc_path_state }}:/state"
    env:
      EXTRA_VALIDATORS: "{{ (groups['bcpeers'] | length) - (groups['bcboot'] | length) }}"
      EXTRA_ENDOWED: "{{ bc_polkadot_extra_endowed }}"
    stop_timeout: 600
    pull: true

- name: "Check health producer"
  uri:
    url: "http://localhost:9933"
    method: POST
    body: '{"jsonrpc":"2.0", "method":"system_health", "params": [], "id":1}'
    body_format: json
    status_code: 200
  register: check_producer
  retries: 15
  delay: 10
  until: check_producer is success
  tags:
    - prod
```



Состав решения: benchmark part

- базируется на npm-пакете **tank.bench-common**, который:
    - многопоточно шлет транзакции с заданным tps (обычно используется JSON RPC или WebSockets)
    - посылает в prometheus основные метрики из бенчмарка
- дополняется написанным под конкретный тест кодом **tank.bench-***, который:
    - отправляет нужный вид транзакций
    - проводит setup в блокчейне (создает аккаунты и их начальные state'ы)
- выбран JS, т.к. почти все блокчейны имеют клиентский код на нём

```
const commitTransaction = async (
    {constructData}:
        CommitTransactionArgs<ReturnType<typeof prepare>, ReturnType<typeof constructBench>>):
    Promise<TransactionResult> => {

    const {usersConfig, keyPairs, userNoncesArray, api} = constructData;

    let senderSeed = getRandomSenderSeed(usersConfig);
    let senderKeyPair = keyPairs.get(senderSeed)!;

    let nonce = Atomics.add(userNoncesArray, senderSeed - usersConfig.firstSeed, 1);

    let receiverSeed = getRandomReceiverSeed(usersConfig, senderSeed);
    let receiverKeyringPair = keyPairs.get(receiverSeed)!;

    let transfer = api.tx.balances.transfer(receiverKeyringPair.address, TOKENS_TO_SEND);
    await transfer.signAndSend(senderKeyPair, {nonce});

    return {code: 10, error: null}
};
```



Результаты

- сделали несколько тестов для собственного алгоритма финализации [RANDPA]
- доказали работоспособность консенсуса в разных конфигурациях
  - проверили со 101 валидатором, geo, packetloss
  - до запуска testnet
- запустили mainnet, с ноября 2019 в production, все работает
- провели [публичное] тестирование сети Polkadot (Parity Substrate)
  - со 101 валидатором, geo и packetloss





Обещали экономию?

- блокчейн-ноды довольно требовательны к ресурсам
  - почти всегда это large-инстансы
- важное отличие от традиционных сервисов
  - потребляют CPU, сеть и диск даже в idle-режиме
- держать testnet для тестов месяцами - дорого
- примерный расчет — на картинке

- Стоимость large-машин составляет $80/mon = $0,11905/h = $0,0019842/min
- Стоимость monitoring-машин составляет $20/mon = $0,02976/h = $0,000496/min
- Стоимость small-машин составляет $20/mon = $0,02976/h = $0,000496/min

Расчет стоимости запусков тестов:

- Тест 1: ((0,000496 * 1) + (0,0019842 * 21)) * 23 = 0,9697766$
- Тест 2: ((0,000496 * 1) + (0,000496 * 21)) * 120 = 1,30944$
- Тест 3: ((0,000496 * 1) + (0,0019842 * 21)) * 43 = 1,8130606$
- Тест 4: ((0,000496 * 1) + (0,0019842 * 21)) * 50 = 2,10821$
- Тест 5: ((0,000496 * 1) + (0,0019842 * 21)) * 90 = 3,794778$
- Тест 6: ((0,000496 * 1) + (0,0019842 * 21)) * 33 = 1,3914186$
- Тест 7: ((0,000496 * 1) + (0,0019842 * 55)) * 108 = 11,839716$
- Тест 8: ((0,000496 * 1) + (0,0019842 * 101)) * 78 = 15,6702156$
- Тест 9: ((0,000496 * 1) + (0,0019842 * 101)) * 75 = 15,067515$
- Тест 10:((0,000496 * 1) + (0,0019842 * 91)) * 101 = 18,2868782$

Сумма: 72.25$



Как использовать в своем блокчейне/БД

- сделать репозитарий с JS-функциями setup’а аккаунтов и отправки одной транзакции (можно по аналогии с tank.bench-polkadot)
- сделать репозитарий с ansible-ролью для provisioning’а кластера БД или блокчейн-сети (можно по аналогии с tank.ansible-polkadot)
  - для блокчейнов на том же ядре или апдейта версии обычно нужно просто поменять URL репозитория или docker image
- *pip3 install mixbytes-tank*
- configure <testcase file>.yml
  - ansible repo, число инстансов, тип облака и API key, Grafana credentials, geo parameters, etc...
- *tank cluster (deploy|run|bench|destroy|inspect|...) <testcase file>.yml*



Спасибо

